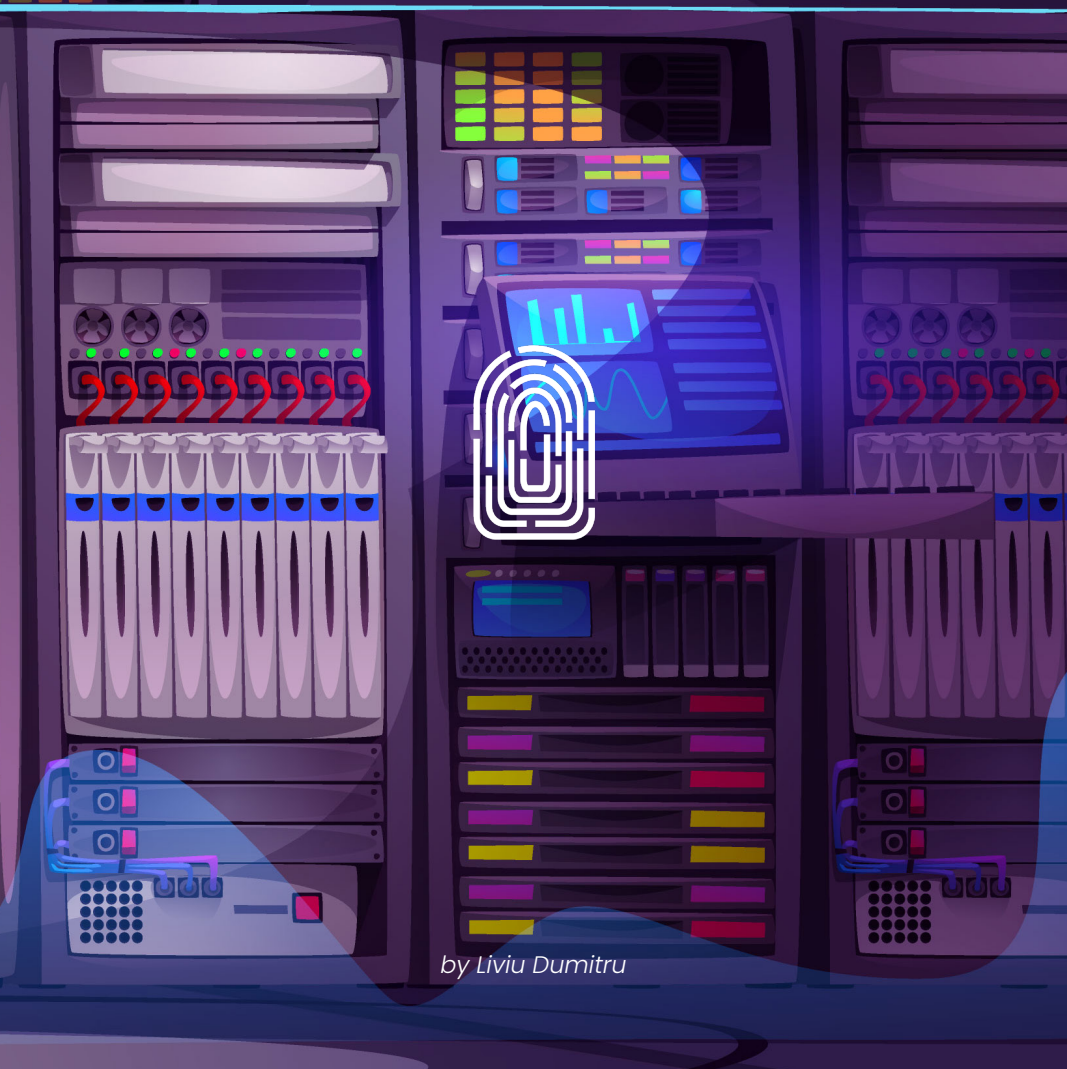
A password is not enough to protect your account

The need for 2FA

TWO FACTOR AUTHENTICATION



by Liviu Dumitru

Dedication

To my wife, children, and parents.

Contents

Chapter 1 - Introduction 6

1.1 - The need of Two Factor Authentication 6
1.2 - Benefits of adding 2FA to your website 9
1.3 - Project Pros and Cons 12
1.4 - Security principles of the application 14

Chapter 2 - Developing the Application 16

2.1 - Application preview on the whole 16
2.2 - Generate the Ruby on Rails application 22
2.3 - Application configuration 24
2.4 - Build authentication system 28
2.5 - Scaffolding CRUD resources 36
2.6 - The models' logic 40
2.7 - OTP controller and background worker 50
2.8 - Messages delivery services 56
2.9 - Administration area 64
2.10 - Project's files and folders structure 67

Chapter 3 - Using the Service 69

3.1 - Forever free web application 69
3.2 - Open-source - Deploy in your own way 76
3.3 - Developing new features 77
Endnotes 78
Information for developer 79

About the author

Liviu Dumitru, 43, is a programmer from Bucharest, Romania, who is currently detained in the state of Kentucky, awaiting trial in a case where he is accused of wire fraud and money laundering. An IT professional with 20 years of experience, a young entrepreneur involved in multiple businesses, a devoted family man married for over 15 years, and the father of two daughters, a well-integrated member of society, Liviu made a mistake a few years ago by creating a software that was subsequently used by others to defraud individuals. Through this book, Liviu wishes to express his desire to make amends for the mistake he committed as much as possible by offering for free his knowledge and work capabilities in the service of community.

A brief statement from Liviu:

First of all, I want to acknowledge my mistake, express my regret, and state that I am willing to do everything in my power to try to make amends and assist the people harmed by the software I created.

My promise, through this public appeal, is to dedicate myself entirely to creating solutions and online services aimed at fighting cybercrime and online identity theft, to help protect people from being harmed or attacked in the online environment. No matter the path I must take in the coming period, this has become my goal: to create something beneficial that helps contribute to the security of people and makes their lives safer online.

I want to sincerely apologize to those who were harmed by the software I created, as well as to my children, my wife, my parents, and my friends, who stand by me during these trying times.

Liviu can be contacted through helpliviu.org for any free assistance or programming topic.

Acknowledgments

There are many people who made this book possible and contributed to it.

My wife Corina provided me invaluable and detailed assistance in all the areas of the book, offering me online resources via Combined Public Communication messaging system that we have in jail, answering me the questions to make the book better from start to finish.

Catalin, one of my best friends from high-school, and the one that guided me into the programming world back in the 1995, helped me converting the paper writing code into digital format, setting up the repository of the open-source project and mounting the application that will be available for free at Secure-2FA.com.

Cosmin, also a good friend from childhood, the CEO of many IT companies, gave extensive help in building helpliviu.org website, where I made my appeal public.

Special thanks to my lawyer Joe, who agreed to mediate the email correspondence of this book content written on papers to my family back home in Romania.

Finally, none of this would be possible or worthwhile without the support of my entire family, and my deepest thanks go to them.

Liviu Dumitru / Kentucky, USA / March 2025

Chapter 1 - Introduction

1.1 - The need of Two Factor Authentication

The usual ways your online accounts gets compromised are by falling a victim of a phishing scheme or if your device gets infected by a malware script.

Phishing

Even if mostly email hosting providers have their own anti-spam filters, the users' email boxes got hit by unsolicited emails, attackers always finding methods to bang into "inbox" folder, passing the spam filters. Through the use of phishing messages, criminals acquire accounts login information from victims, like username and password.

The scenario is as follows:

A criminal sends a fake message claiming to be from an institution or business which the prospective consumer to validate his account information by clicking on a link provided in the message. The individuals are then sent to a fraudulent website that resembles the actual institution or business and are asked to enter their login information.

Malware

Another way that criminals use to get accounts' credentials is by spreading to internet malicious programs, called "malware", that infects your device, no matter its type, laptop, tablet, or smartphone. Once the device is compromised, the malware sends all of your browser's saved credentials to their server and you'll not even notice this action. Data is then used by criminals to engage in fraud or resold to others through dark web markets.

Computer security experts continue to express concerns about the actual number of malicious software scripts and the increases they expect to see in the future; the number of new malware introduced into the wild each year is tremendous.

Conclusion

Protecting users' accounts and their sensitive information only with a password can be a compromise.

As a developer, you should consider protecting the authentication sections with a second layer of security. I'm sure that 2FA term is familiar to you. Using the service Secure-2FA.com for free or installing the GitHub repository github.com/helpliviu/secure-2fa in your own IT infrastructure, you can quickly realize a 2FA setup.

You'll find in the next section the benefits of 2FA for websites owners and also for their users.

1.2 - Benefits of adding 2FA to your website

Implementing Two Factor Authentication (2FA) using a One-Time Password (OTP) sent via email or SMS brings significant advantages for both businesses and users in terms of security, trust and user experience. Below is a detailed breakdown of the benefits:

For the Businesses

Enhanced Security & Data Protection

- Adds an extra layer of security beyond just the password, reducing the risk of account takeovers.
- Protects sensitive business and user data from unauthorized access.
- Helps mitigate the impact of password leaks (if user reuse passwords across platforms).

Compliance with Regulations

- Helps meet GDPR, ISO27001 or PCI-DSS security standards that often require multi-factor authentication.
- Builds a solid foundation for data protection laws compliance.

User Trust & Reputation

- Demonstrates to users that the business takes security seriously, which improves trust and brand reputation.
- Reduces negative PR or legal risks from data breaches.

Fraud Prevention

- Protects against credential stuffing attacks, phishing, and brute-force attacks.
- OTP codes make it harder for attackers to access accounts, even if they have the password.

User Activity Monitoring

- Allows businesses to track suspicious login attempts and flag accounts with multiple failed OTP attempts.

Self-Service Security

- Easier for businesses to reset access and verify users without manual intervention or customer support.

For the User

- Increased Account Security - Users feel safer knowing that even if their password is stolen, their account is protected by a second layer of authentication.
- Convenience - Most users have access to email on their smartphones, making OTP codes easily accessible without needing extra apps.
- No Additional App Required - Unlike authenticator apps, email or SMS OTP doesn't require installing third-party apps, making it more user-friendly.
- Prevention of Unauthorized Logins - Even if someone knows the user's password, they still need the OTP from the user's email or SMS to access the account.
- Transparent Process - Easy to understand verification process without technical jargon.
- Notifications of Suspicious Activity - If a user receives an OTP they didn't request, they can immediately identify suspicious activity.

- Password Reset Verification - OTP codes sent to email or SMS are a common method of verifying identity during password resets.
- User Control - Gives users control over their account access, knowing they are always required to authorize logins.

Conclusion

Implementing 2FA via email or SMS messages OTP is a win-win solution for both businesses and users. It significantly enhances security, improves user trust, and helps businesses meet compliance regulations - all without adding major complexity. However, businesses should always encourage users to choose OTP + strong passwords rather than relying on just OTP alone for maximum protection.

1.3 - Project Pros and Cons

This 2FA project addresses a real need for developers looking to integrate OTP-based authentication without being locked into a single provider and offering customizable setup out of the box.

Pros

- Open-Source & Free to Use - I consider this a major advantage since most solutions charge for both service usage and OTP delivery.

- Flexible Delivery Options - Allowing users to select from multiple third-party providers (Postmark, Twilio, Sendgrid, etc) or use their SMTP credentials provides cost control.

- Simple API Integration - Only two API calls (generate OTP and verify OTP) make adoption easy.

- Multi-Language Support - This feature is crucial for international applications.

- Scalability - Supporting multiple projects under one account and handling high volumes efficiently is a strong point.

- Security Enhancements - Encryption and IP/User-Agent validation increase security against attacks.

- Self-Delivery Option - Advanced users can generate OTPs but send them via their own email/SMS infrastructure.

- Customizable Templates - Branding consistency is ensured.
- Admin Dashboard - A simple and intuitive UI for managing settings, templates, and logs.
- Analytics & Logs - Provide insights into OTP request trends and failures.
- Rate Limiting - Prevent brute force attacks on OTP verification.
- Custom Expiry Times - Allow developers to specify OTP expiration periods dynamically.
- Webhook Notifications - Let websites receive real-time updates on OTP verification events.

Cons

- Limited to OTP-based 2FA - No support for hardware keys (like Yubikey), biometric authentication, or time-based authenticator apps (Google Authenticator).
- Potential Deliverability Issues - Relying on third-party services means you might face rate limits, email blacklisting, or SMS filtering.

1.4 - Security principles of the application

Framework Security

Ruby on Rails (RoR) is widely regarded as a secure web application framework when used correctly. It provides numerous built-in features to mitigate common web vulnerabilities.

Here's a breakdown of RoR's security features active in our application as well as some advice and common vulnerabilities if you intend to extend the application features.

Cross-Site Scripting (XSS) Protection - Rails automatically escapes HTML in views by default, preventing XSS attacks.

- Cross-Site Request Forgery (CSRF): Rails provides protection by default using a CSRF-token that is validated for every form submission.
- SQL Injection Protection: ActiveRecord automatically escapes query parameters, mitigating SQL injection risks.
- Mass Assignment Protection: Strong parameters ensure only explicitly permitted attributes are accepted during model creation or updates.
- Content Security Policy: Rails supports CSP to limit sources for scripts and styles, reducing XSS risks.
- Encryption and Secrets Management: Rails uses ActiveSupport::MessageEncryptor for securely encrypting sensitive data, including configuration values or database fields.

While Rails is secure by default, improper usage or misconfigurations can still introduce vulnerabilities. If you intend to fork the repository to extend application features, you should be careful about the following:

- Insecure Gems: Using outdated or poorly maintained gems can introduce vulnerabilities.
- Improper Configuration: Misconfigured server settings or deployment environments can expose sensitive data.
- Lack of Validation: Failing to validate input data properly can lead to application-specific vulnerabilities.

Data encryption

Application uses encryption for all stored and transmitted data over API usage calls. All sensitive fields are encrypted before stored in the database, thanks to ActiveRecord Encryption found in the latest feature of Rails 7 that supports natively encrypted attributes directly within the model layer, and PostgreSQL's built-in capabilities (like JSONB columns) that can handle serialized data structures like hashes.

Communication data passed to and from the API is encrypted using AES-256 (Advanced Encryption Standard) with a symmetric 32 bytes key, as it's a widely supported and secure encryption method for mostly programming languages.

Secure access

Access to the application backend is protected by a second layer of security using TOTP codes generated by time-based authenticator applications like Google Authenticator or Twilio Authy. Backup codes are provided for cases when access to authenticator application is not possible.

Chapter 2 - Developing the Application

2.1 - Application preview on the whole

I'll break the application architecture into modular components to better understand it.

Technologies used

***Framework*:** Ruby on Rails (version 7) for backend and API
***Database*:** PostgreSQL with encryption plugins.
***Processing engine*:** Redis and Sidekiq for handling queued tasks.
***Security*:** Devise and TOTP for authentication; OpenSSL for encryption.

Backend sections

Authentication in application will be made by model ***Client***. The whole register and sign-in process will be handled by the *Devise gem*, developed by *collective idea* team, a stable and very well documented gem. We'll add the devise-two-factor gem to the default configuration of Devise, to cover a second layer of security, using TOTP codes.

The backend will consist of the following sections:

Projects

Management of the websites' sections where you want to implement 2FA. Every project has its unique *Users* and every user has its *Verifications*, in fact, the OTP codes.

Channels

Setup how messages, of types emails or SMS, will be delivered.

Templates

Define the content of the OTP messages generated by the application. Each template has its own *Template*

Versions, based on multiple languages.

Logs
A history of OTP generation and verification calls to the API.

A complete list of application's models and their attributes on following sections.

API design
The API consists of two calls, one to generate the user's OTP and the other one to verify it.

Endpoints:
POST `/otp/generate` - it will receive the encrypted data from the website, will validate it, generate and deliver the template containing the OTP code

POST `/otp/verify` - it will receive the encrypted data with OTP verification details

You'll find more details on how to use the API at **section 3.1.**

Models' attributes and relationship diagram a.k.a. schema.db

CLIENT

```
email: string REIU
password: string
password_confirmation: string
confirmation_token: string
confirmed_at: datetime
confirmation_sent_at: datetime
unconfirmed_email: string
otp_secret: string E
otp_required_for_login: boolean: false
otp_backup_codes: text E
admin: boolean: false
```

PROJECT

```
client: reference R
title: string Ri
public_key: string EIU
secret_key: string E
channel: reference R
template: reference R
expiration_seconds: integer: 60
max_attempts: integer: 5
webhook_url: string E
```

USER

```
project: reference R
uuid: string REI
```

LOG

```
client: reference R
project: reference
user: reference
message: text R
level: string I
```

CHANNEL

```
client: reference R
title: string Ri
method: string R
provider: string R
credentials: text RE
```

TEMPLATE

```
client: reference RI
title: string Ri
markup: string R
```

TEMPLATE VERSION

```
template: reference R
subject: string
body: text R
lang: string: en Ri
```

VERIFICATION

```
user: reference R
otp_code: string E
otp_sent_at: datetime I
attempts: integer: 0
user_agent: text E
ip_address: string E
to: string REI
lang: string
template_request: boolean: false
delivery_identifier: string
delivery_status: string: not-known I
```

LEGEND

R - required
i - index
U - unique
E - encrypted

NOTE: For each encrypted field, a new field must be added, with the same name and suffixing in _iv, with type string, no matter the encrypted field type (ex: uuid_iv).

2.2 - Generate the Ruby on Rails application

First things first, we need to create the application.

The prerequisites
Verify that the right versions of Ruby and Rails are installed. You'll need at least Ruby 3.1 and Rails 7. If you've got older version of Rails installed, you can install Rails 7 using:

```
$ gem install rails -v 7
```

Node.js v.18 or newer and Yarn are required to transpile Javascript and install backend libraries. Get Node.js from https://nodejs.org/en/download. Install Yarn using:

```
$ npm install --global yarn
```

Ensure that you have PostgreSQL and Foreman installed. Foreman can be installed by running:

```
$ gem install foreman
```

Create the Rails application
Run the command below to create the Rails application

```
$ rails new secure-2fa -j esbuild --css -d
postgresql
```

This will create the folder named “secure-2fa”. Now you can start the application using:

```
$ cd secure-2fa
$ bin rails db:prepare
$ bin/dev
```

The Rails server should now be running and you’ll see the page at http://localhost:3000.

2.3 - Application configuration

Required gems
Add the following gems to the Gemfile.

```
#/Gemfile
    gem 'devise' # Authentication system
    gem 'devise-two-factor' # Devise addon for
TOTP authenticator
    gem 'rqrcode' # QR code generator needed to
link authenticator applications
    gem 'sidekiq' # Background jobs processing
    gem 'redis'
    gem 'dotenv-rails' # Storing environment
variables
    gem 'ffi-libphonenumber' # Google's library to
validate phone formats
# gems for email/sms delivery services
    gem 'twilio-ruby'
    gem 'mailjet'
    gem 'plivo'
    gem 'sendgrid-ruby'
```

Install gems by running:

```
    $ bundle install
```

Configure Sidekiq

Be sure that you have started an instance of Redis. Configure Sidekiq as follows:

```
#/config/initializers/sidekiq.rb
Sidekiq.configure_server do |config|
config.redis = url: 'redis://localhost:6379/0'
end

Sidekiq.configure_client do |config|
config.redis = url: 'redis://localhost:6379/0'
end
```

Update Rails to include Sidekiq for background jobs processing.

```
#/config/application.rb
    config.active_job.queue_adapter = :sidekiq
```

Include sidekiq to start with Foreman command bin/dev. Add the following:

```
#/Procfile
    bundle exec sidekiq
```

Optional - If you want to monitor Sidekiq activity in web browser you have to add the gem "sidekiq-web" to Gemfile and edit the routes as following:

```
#/config/routes.rb
    require 'sidekiq/web'
    mount Sidekiq::Web => '/sidekiq' # protect
route with http basic authentication
```

Configure database encryption
Rails automatically generates a master key and credentials for encryption. Run the following command, then add the content below:

```
$ bin/rails credentials:edit
active_record_encryption:
primary_key:
deterministic_key:
key_derivation_salt:
```

In the desired environment .env file add the following:

```
#/development.env
ENCRYPTION_PRIMARY_KEY = generated_primary_key
ENCRYPTION_DETERMINISTIC_KEY = generated_deter-
ministic_key
ENCRYPTION_KEY_DERIVATION = generated_salt
```

To generate secure keys use the following command:

```
$ bin/rails secret
```

Configure Devise
Initialize Devise with the following command:

```
$ bin/rails generate devise:install
```

. Setup Devise to send asynchronous emails with Sidekiq. Edit Devise initializer file:

```
#/config/initializers/devise.rb
config.mailer = 'Devise::Mailer'
config.async = true
```

Add a "mailer" queue to Sidekiq:

```
#/config/sidekiq.yml
concurrency: 5
queues:
- default
- mailers
```

2.4 - Build authentication system

Setup Devise for "Client" model

As I mentioned earlier on **section 2.1**, "Client" is the entity that will use the application, in fact, you as a developer. Run the following command to generate Devise for Client model:

```
$ bin/rails generate devise Client
```

In the generated migration file, include all missing attributes that belong to Client model found on **section 2.1** (schema.db), then migrate the database using:

```
$ bin/rails db:migrate
```

Generate the following Devise controllers where registration and authentication logic will be changed to add TOTP:

```
$ bin/rails generate devise:controllers clients
-c=registrations confirmations sessions
```

Add the following code to each controller file:

```
#/app/controllers/clients/registrations_controller.rb
class Clients::RegistrationsController
<Devise::RegistrationsController
  def create
    build_resource(signup_params)
    resource.save
    if resource.persisted?
      resource.generate_and_send_confirmation_
instructions
      render json: { message: “Please confirm your
email to complete registration.” }
      else
        render json: { errors: resource.errors.full_
messages,
          status: :unprocessable_entity }
        end
      end
      private
      def signup_params
        params.require(:client).permit(:email)
      end
    end
```

```
#/app/controllers/clients/confirmations_controller.rb
Class Clients::ConfirmationsController < Devise::Con-
afirmationsController
def show
  self.resource = resource_class.confirm_by_
token(params[:confirmation_token])
  if resource.errors.empty?
    redirect_to edit_client_password_path(resource,
confirmation_token: params[:confirmation_token]),
```

```
notice: “Please set your password”
  else
    render json: {errors: resource.errors.full_
messages},
    status: :unprocessable_entity
  end
end
```

```
#/app/controllers/clients/sessions_controller.vb
Class Clients::SessionsController < Devise::Sessions-
Controller
def create
  self.resource = warden.authenticate!(auth_options)
  if resource.otp_required_for_login
    session[:client_id] = resource.id
    redirect_to verify_otp_login_path
  else
    sign_in(resource)
    redirect_to root_path, notice: “Signed in
successfully!”
  end
end
def verify_otp
  @client = Client.find(session[:client_id])
end
def confirm_otp
  @client = Client.find(session[:client_id])
  if @client.validate_and_consume_otp!(params[:otp_
attempt])
  sign_in(@client)
  redirect_to root_path, notice: “Signed in
successfully using 2FA”
else
  if @client.verify_backup_code(params[:otp_attempt3])
```

```
    sign_in(@client)
    redirect_to root_path, notice: “Signed in
successfully using backup code”
  else
    flash[:alert] = “Invalid 2FA or backup code”;
render :verify_otp
  end
end
end
end
```

Custom registration and signin flows

We want the registration to be made only with an email address. A registration link will be sent to email. After visiting the link, the client must set his own password and in the next page he will need to link his authenticator application.

Signin flow will redirect client to a TOTP page after a successful login with email and password. Backup codes are provided if client can’t access his authenticator application.

Use the commands below to generate controllers for both scenarios:

```
    $ bin/rails generate controller clients/passwords
    $ bin/rails generate controller clients/two_fac-
tor_auths
```

Add the following code to each controller file:

```
#/app/controllers/clients/passwords_controller.rb
class Clients::PasswordsController <
ApplicationController
  before_action :authenticate_client!

  def edit; end

  def update
    if current_client.update(password_params)
      current_client.generate_two_factor_secret_if_
needed!
      redirect_to setup_2fa_path
    else
      render :edit
    end
  end

  private
  def password_params
    params.require(:client).permit(:password,
:password_confirmation)
  end
end
```

```
#/app/controllers/clients/two_factor_auths_controller.
rb

class Clients::TwoFactorAuthsController <
ApplicationController
  before_action :authenticate_client!

  def new
    @qr_code = RQRCode::QRCode.new(current_client.
```

```
provision_uri).as_png(size: 200).to_data_url
  end

  def verify
    if current_client.validate_and_consume_
otp!(params[:otp_attempt])
      current_client.update(otp_required_for_login:
true)
      redirect_to root_path, notice: “2FA successfully
enabled”
    else
      flash[:alert] = “Invalid OTP code”
      render :new
    end
  end

  def generate_backup_codes
    current_client.generate_backup_codes
# respond with modal to show the backup codes once
  end
end
```

Custom routes for Devise setup

Edit routes file and add the following:

```
# /config/routes.rb
devise_for :clients, controllers: {
  registrations: ‘clients/registrations’,
confirmations:
  ‘clients/confirmations’, sessions: ‘clients/sessions’
}

devise_scope :client do
  get ‘clients/password/setup’, to: ‘clients/
passwords#edit’, as: :edit_client_password
patch ‘clients/password/setup’, to: ‘clients/
```

```
passwords#update'
  get 'setup_2fa', to: 'clients/two_factor_auths#new'
  post 'verify_otp', to: 'clients/two_factor_
auths#verify'
  get 'generate_backup_codes', to: 'clients/two_
factor_auths#generate_backup_codes'
  get 'verify_otp_login', to: 'clients/
sessions#verify_otp'
  post 'verify_otp_login', to: 'clients/
sessions#confirm_otp'
end
```

Model configuration for Devise

Add the following code to Client model:

```
# /app/models/client.rb
class Client < ApplicationRecord
  devise :database_authenticatable, :registerable,
:recoverable,
  :rememberable, :validatable,
  :confirmable, :two_factor_authenticatable, :otp_
secret_
  encryption_key => ENV['OTP_SECRET_ENCRYPTION_KEY']
  def generate_two_factor_secret_if_needed!
    self.otp_secret ||= Client.generate_otp_secret
  end
  def generate_and_send_confirmation_instructions
    generate_confirmation_token! if confirmation_token.
    blank?
    send_confirmation_instructions
  end
  def provision_uri
label = "#{ENV['APP_NAME']}: #{email}"
issuer = ENV['APP_NAME']
```

```
otp_provisioning_uri(label, issuer: issuer)
end
serialize :otp_backup_codes, Array
def generate_backup_codes
  self.otp_backup_codes = 10.times.map { SecureRandom.
    hex(3).upcase }
    save
  end
  def verify_backup_code(code)
    if otp_backup_codes.include?(code)
      self.otp.backup_codes.delete(code)
      save
      true
    else
      false
    end
  end
end
```

2.5 - Scaffolding CRUD resources

Considering my current situation (see "**About the author**" section), I will use Rails scaffold generator for backend sections as there are minor changes to each CRUD controller logic.

Complete each scaffold command with all model's attributes from **section 2.1**. Don't forget about the additional string field required by each encrypted field (ex: public-key requires public-key-iv...).

```
$ bin/rails generate scaffold Project
client:reference title:string...

$ bin/rails generate scaffold Channel
client:reference title:string...

$ bin/rails generate scaffold Template
client:reference title:string...
```

For each of these three controllers you'll have to do the following two tasks. I'll give example for Projects controller and you must do the same for Channels and Templates controllers.

1. Restrict controller access only for logged in clients by adding the following line at the start of controller:

```
before_action :authenticate_client!
```

2. Replace any occurrence of **Project** with **current_client.projects**

After changes, the controller should look like this:

```
#/app/controllers/projects_controller.rb
class ProjectsController < ApplicationController
  before_action :authenticate_client!
  def index
    @projects = current_client.projects.all.page
    params[:page]
  end
  def new
    @project = current_client.projects.new
  end
  …
  …
end
```

Note for Templates controller - TemplateVersion's nested attributes needs to be added to "template_params" declaration found at the end of the controller, after the last template's attribute, like this:

```
  …:markup, template_version_attributes: [:id,
:subject, :body, :lang]
```

Generate the rest of models

The other application's models are as follows:

- The *User* model, that belongs to *Project*. It stores the users' information from the websites the service is implemented.

- The *Verification* model, that belongs to *User*. It stores the information of generated OTP codes and their verification attributes.

- The *TemplateVersion* model, that belongs to *Template*. It stores the multi-language versions of the OTP messages sent to the users.

- The *Log* model. It stores the history of users' verifications.

As we don't need controllers and views for them, we'll generate only the models files. Complete each model's attributes from **section 2.1**.

```
$ bin/rails generate model User project:reference
uuid:string uuid-iv:string

$ bin/rails generate model Verification user:reference
otp_code:string otp-code-iv:string…

$ bin/rails generate model TemplateVersion
template:reference subject:string body:text…

$ bin/rails generate model Log client:reference
project:reference user:reference…
```

Migrate database changes

It's important to update each model's migration with required, unique, or indexes statements as mentioned in **section 2.1** - schema.rb. Below is an example for projects migration's file.

```
#/db/migrations/xxxxxxx_create_projects.rb

create_table :projects do |t|
t.references :client, null: false, foreign_key: true
t.references :channel, null: false, foreign_key: true
```

```
t.references :template, null: false, foreign_key: true
t.string :title, null: false
t.string :public_key
t.string :public_key_iv
t.string :secret_key
t.string :secret_key_iv
t.integer :expiration_seconds, default: 60
t.integer :max_attempts, default: 5
t.string :webhook_url
t.string :webhook_url_iv
end

add_index :projects, :title
add_index :projects, :public_key, unique: true
```

Now you can run the database migration with the following command:

```
$ bin/rails db:migrate
```

Routes check

The scaffold's commands also populate the routes file with specific resources.

```
#/config/routes.rb
resources :projects
resources :channels
resources :templates
```

2.6 - The models' logic

First of all, we have to create a model concern that we'll use in Channel and Template.

```
#/app/models/concerns/check_linked_projects.rb
module CheckLinkedProjects
  extend ActiveSupport::Concern
  included do
    before_destroy do
      if self.projects.count > 0
        errors.add("Resource is linked to projects")
        throw(:abort)
      end
    end
  end
end
```

As we have all the application's models in place, we have to pass each file and add model's logic, representing validations, relations, nested attributes, delegate fields, encrypted fields, callbacks, scopes and helper functions.

```
#/app/models/client.rb
# Devise logic from section 2.4
validates :email, presence: true, email: true,
uniqueness: true
has_many :projects, dependent: :destroy
has_many :channels, dependent: :destroy
has_many :templates, dependent: :destroy
has_many :logs, dependent: :destroy
encrypts :email, deterministic: true
encrypts :otp_secret, deterministic: true
encrypts :otp_backup_codes, type: :array
```

```
default_scope -> { order(id: :desc) }
before_create :setup_admin
def setup_admin
self.admin = true if Client.count == 0
end
```

```
#/app/models/project.rb
class Project < ApplicationRecord
   validates :title, presence: true
   validates :expiration_seconds, numericality: {true,
greater_than: 50}
   validates :max_attempts, numericality: {true,
greater_than: 0}
   validates :webhook_url, allow_blank: true, format:
{with: URI::DEFAULT_PARSER.make_regexp, message: "%s
must be a valid URL"}
   belongs_to :client
   belongs_to :channel
   belongs_to :template
   has_many :users, dependent: :destroy
   has_many :logs, dependent: :destroy
   accept_nested_attributes_for :channel
   accept_nested_attributes_for :template
   encrypts :public_key, deterministic: true
   encrypts :secret_key
   encrypts :webhook_url
   default_scope -> { order(:title) }
   scope :deliveries_by -> (method) {
   includes(:channel).where("channel.method = ?",
method) }
   before_create :generate_keys
   def generate_keys
self.public_key = SecureRandom.hex(16)
   self.secret_key = SecureRandom.hex(16)
```

```
end

end
```

```
#/app/models/user.rb
class User < ApplicationRecord
   belongs_to :project
   has_many :verifications, dependent: :destroy
   has_many :logs, dependent: :destroy
   validates :uuid, presence: true, uniqueness: {true,
scope: :project_id}
   delegate :client, :channel, :template, to: :project
   encrypts :uuid, deterministic: true
   default_scope -> { order(:uuid) }
end
```

```
#/app/models/verification.rb
class Verification < ApplicationRecord
  validates :to, presence: true
  validates :delivery_status, inclusion: { in: self.
delivery_statuses, message: “Invalid delivery status.”
  }

  belongs_to :user
  delegate :client, :project, :channel, :template, to:
:user
  encrypts :to, deterministic: true
  encrypts :otp_code
  encrypts :user_agent
  encrypts :ip_address
  default_scope { order(otp_sent_at: :desc) }
  before_create :validate_to, :generate_otp
  after_create :deliver_otp
```

```
  def validate_to
    if self.to.include?("@")
      begin
        parsed = Mail::Address.new(self.to)
        parsed.address == self.to AND parsed.domain
AND parsed.domain.include?('.')
      rescue
        errors.add("Invalid email format");
throw(:abort)
      end
    else
      begin
        parsed = FFI::LibPhoneNumber.parse(self.to,
nil)
        self.to = parsed.international_format if
parsed&.valid?
      rescue
        errors.add("Invalid phone format")
        throw(:abort)
      end
    end
  end

  def generate_otp
    self.otp_code = rand(100_000..999_999).to_s
    self.otp_sent_at = Time.current
  end

  def deliver_otp
    DeliverOtp.perform_later(self) if self.template_
request == false
  end

  def self.delivery_statuses
    %w['not-known', 'delivered', 'not_delivered']
  end
```

```
  def generate_response
    response = {
      success: true, message: “OTP generated
successfully.”,
      subject: self.template_request ? self.subject :
nil,
      body: self.template_request ? self.body : nil
    }; return response
  end

  def expired?
    otp_sent_at < self.project.expiration_seconds.
seconds.ago
  end

  def exceeded_attempts?
    attempts >= self.project.max_attempts
  end

  def unauthorized_device?(user_agent, ip_address)
    if self.user_agent.present? && self.ip_address.
present?
      if self.user_agent == user_agent && self.ip_
address == ip_address
        false
      else
        true
      end
    else
      false
    end
  end

  def subject
    (self.template.template_versions.find_by(lang:
self.lang) ||
      self.template.template_versions.first).subject
```

```
  end

  def body
    (self.template.template_versions.find_by(lang:
self.lang) || self.template.template_versions.first).
body.gsub("{{OTP_CODE}}", self.otp_code)
  end
end
```

```
#/app/models/channel.rb
class Channel < ApplicationRecord
  include CheckLinkedProjects
  validates :title, presence: true
  validates :method, presence: true, inclusion: { in:
    self.delivery_methods, message: "%{value} is not a
    valid method" }
    validates :provider, presence: true, inclusion: {
in: self.providers_handles, message: "%{value} is not
a valid provider" }
      validates :credentials, presence: true
      serialize :credentials, Hash
      belongs_to :client
      has_many :projects, dependent: :nullify
      encrypts :credentials, type: :json
      default_scope -> { order(:title) }
      before_save :credentials_to_json

   def credentials_to_json
        self.credentials = self.credentials.to_json
   end

   def self.delivery_methods
        %w[email, sms]
   end
```

```
    def self.providers_handles
         self.providers.map { |provider|
provider[:handle] }
    end

    def self.providers_by_method(method)
         self.providers.map { |provider|
[provider[:name], provider[:handle]] } if
provider[:methods].
           include?(method)
    end

    def providers
     providers = [
      { name: “Postmark”, handle: “postmark”,
methods:[“email”], credentials: { email: [“server_
token”,“from_email”] } },
      { name: “Sendgrid”, handle: “sendgrid”,
methods:[“email”], credentials: { email: [“api_key”,
“from_email”] } },
      { name: “Clicksend”, handle: “clicksend”,
methods:[“email”, “sms”], credentials: { email:
[“username”,“api_key”, “from_email”], sms:
[“username”, “api_key”,“fendev_id”] } },
      { name: “Mailjet”, handle: “mailjet”,
methods:[“email”], credentials: { email: [“api_key”,
“secret_key”, “from_email”, “from_name”] } },
      { name: “Sparkpost”, handle: “sparkpost”,
methods:[“email”], credentials: { email: [“api_key”,
“from_email”] } },
      { name: “Elasticmail”, handle: “elasticmail”,
methods:[“email”], credentials: { email: [“api_key”,
“from_email”] } },
      { name: “Twilio”, handle: “twilio”, methods:
[“sms”],credentials: { sms: [“account_sid”, “auth_
token”,“from”] } },
```

```
      { name: “Plivo”, handle: “plivo”, methods:
[“sms”],credentials: { sms: [“auth_id”, “auth_token”,
“from_sender_id”] } },
      { name: “SMTP”, handle: “smtp”, methods:
[“email”],credentials: { email: [“address”, “port”,
“username”,“password”, “from_email”, “authentication”,
“enable_starttls_auto”, “ssl”, “tls”, “verify_mode”] }
}
  ]
      return providers
    end
  end
```

```
#/app/models/template.rb
class Template < ApplicationRecord
  include CheckLinkedProjects
  validates :title, presence: true
  validates :markup, presence: true, inclusion: { in:
    Template.body_markups, message: “%{value} is not a
    valid markup” }
    belongs_to :client
    has_many :projects, dependent: :nullify
    has_many :template_versions, dependent: :destroy
    accept_nested_attributes_for :template_versions
    default_scope -> { order(:title) }

    def body_markups
      %w[html, text]
    end
  end
```

```
#/app/models/template_version.rb
class TemplateVersion < ApplicationRecord
  validates :lang, presence: true, inclusion: { in:
    TemplateVersion.languages, message: “%{value} is
not a valid language” }
    validates :subject, presence: true, if: -> {
      template&.markup == “html” }
      validates :body, presence: true, format: { with:
/OTP_CODE/, message: “%{value} must include variable
OTP_CODE” }
      belongs_to :template
      default_scope -> { order(:lang) }

      def self.languages
        %w[en es fr de it jp nl hu ro …]
      end
    end
```

```
#/app/models/log.rb
class Log < ApplicationRecord
  validates :message, presence: true
  validates :level, presence: true, inclusion: { in:
    Log.levels }
    belongs_to :client
    belongs_to :project
    belongs_to :user

    default_scope -> { order(id: :desc) }
    before_create :setup_relations

   def setup_relations
      self.project_id = self.user.project.id
      self.client_id = self.user.project.client.id
   end
```

```
  scope :filter_by_levels -> (levels_array) {
  where(“level IN ?”, levels_array) }

def self.levels
    %w[success, info, warning, danger]
end
```

2.7 - OTP controller and background worker

The controller logic

Run the following command to create the controller that generate and verify the OTP codes

```
$ bin/rails generate controller otp generate verify
```

Populate the controller with the following code:

```
#/app/controllers/otp_controller.rb
class OtpController < ApplicationController
  require “openssl”
  require ‘base64’

  before_action :find_project

  def generate
    uuid, to = decrypt(params[:encrypted_data], @
project.secret_key).split(‘#’)
    @user = @project.users.find_or_create_by(uuid:
uuid)

  if @user.persisted?
   @user.verifications.where(“otp_sent_at < ?”,
    ENV[‘OTP_STORE_MINS’].to_i.minutes.ago).destroy_
all
   verification = @user.verifications.create!(to:
to, user_agent: params[:user_agent], ip_address:
params[:ip_address], lang: params[:lang], template_
request: params[:template_request])

  if verification.persisted?
```

```
    @json_response = verification.generate_response
    @log_level = :info
    render json: @json_response
  else
    @json_response = { success: false, message: “Error
generating OTP” }
      @log_level = :danger
      render json: @json_response, status:
:unprocessable_entity
    end

    @user.logs.create(message: @json_
response[:message],
      level: @log_level.to_s)
  end
end

  def verify
  uuid, otp = decrypt(params[:encrypted_data], @
project.secret_key).split(‘#’)
@user = @project.users.find_by(uuid: uuid)

  if @user
  verification = @user.verifications.first
  if verification.nil? || verification.expired?
    @json_response = { verified: false, message:
“Invalid or expired OTP” }
      @log_level = :warning
      render json: @json_response, status:
:unauthorized
    elsif verification.exceeded_attempts?
      @json_response = { verified: false, message: “Too
many requests. Request a new OTP” }
        @log_level = :danger
        render json: @json_response, status: :too_
```

```
many_requests
      elsif verification.otp_code == otp
        if verification.unauthorized_
device?(params[:user_agent], params[:ip_address])
        @json_response = { verified: false, message:
          “Unauthorized device.” }
          @log_level = :danger
          render json: @json_response, status:
:unauthorized_device

        else
          verification.destroy
          @json_response = { verified: true, message:
“OTP verified successfully” }
            @log_level = :success
            render json: @json_response
        end

        else
          verification.increment!(:attempts)
          @json_response = { verified: false, message:
“Invalid OTP verification” }
            @log_level = :warning
            render json: @json_response, status:
:unauthorized
        end

          @user.logs.create(message: @json_response
[:message], level: @log_level.to_s)
       WebhookCall.perform_later(@project.webhook_url,
params[:encrypted_data],
      @json_response) if @project.webhook_url.present?
        end
      end
  protected
```

```
      def decrypt(encrypted_data, key)
        encoded = Base64.decode64(encrypted_data)
        iv = encoded[0..15]
        encrypted = encoded[16..-1]
        decipher = OpenSSL::Cipher::AES256.new(:CBC)
        decipher.decrypt
        decipher.key = key
        decipher.iv = iv
        decipher.update(encrypted) + decipher.final
  end

  private
      def find_project
        @project = Project.find_by(public_key:
params[:public_key])
        render json: { success: false, message:
“Project not found” }, status: :not_found unless @
project
        end
      end
```

Add the following routes:

```
#/config/routes.rb
constraints subdomain: ‘api’ do
post ‘otp/generate’, to: ‘otp#generate’
post ‘otp/verify’, to: ‘otp#verify’
end
```

The message delivery job

Run the following command to create a background worker for OTP message delivery:

```
$ bin/rails generate job DeliverOtp
```

Populate the job file with the following code:

```
#/app/jobs/deliver_otp_job.rb
class DeliverOtpJob < ApplicationJob
  queue_as :default

  def perform(verification)
  res = eval("#{verification.channel.provider.
titlecase}Service")

  -send(verification.channel.method, verification.
channel.
  credentials, verification.to, verification.body,
  verification.subject)

  -send_message
  verification.update_attributes delivery_identifier:
  res[:identifier], delivery_status: res[:status]
  end
end
```

Webhook call

At the end of the "verify" function in the OTP controller you notice a Webhook call. If project has defined a "webhook_url", it will receive real-time updates on OTP verification events.

You have to generate a background job for it running the following command:

```
$ bin/rails generate job WebhookCall
```

Add the following code to the job file:

```
#/app/jobs/webhook_call_job.rb
class WebhookCallJob < ApplicationJob
  queue_as :default

  def perform(webhook_url, encrypted_data, json_
response)
  uri = URI.parse(webhook_url)
  http = Net::HTTP::Post.new(uri.path)
  request[‘Content-Type’] = “application/json”

  request.body = {encrypted_data: encrypted_data,
response: json_response}
    http.request(request)
  end
end
```

2.8 - Messages delivery services

The application has 10 built-in services that you can choose to deliver OTP messages by email or SMS. Other delivery providers can be easily implemented. You can find more details about this on **section 3.3**.

Each sending method, including own SMTP, has its own service file. Create the following files and populate its specific code. For the services that had maintained and well documented gems, I choose to use them, as for the other ones, I've used the ruby built-in net/http library to make the service's calls for delivery.

```
#/app/services/postmark.rb
class PostmarkService
  def self.send_message(method, credentials, to,
message_body, message_subject)
  server_token = credentials[:server_token]
  from_email = credentials[:from_email]
  uri = URI.parse('https://api.postmarkapp.com/email')
  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true

  request = Net::HTTP::Post.new(uri.path)
  request["Content-Type"] = "application/json"
  request["X-Postmark-Server-Token"] = server_token
  body = {"From" => from_email, "To" => to, "Subject"
=> message_subject, "HtmlBody"
    => message_body, "TextBody" => message_body.to_
text}.to_json
  request.body = body

  response = http.request(request)
```

```
  # try to get identifier and status from response
body; convert status to our values return
  {identifier: identifier, status: status}
  end
end
```

```
#/app/services/sendgrid_service.rb
class SendgridService
  require 'sendgrid-ruby'
  include SendGrid

  def self.send_message(method, credentials, to,
message_body, message_subject)
  api_key = credentials[:api_key]
  from_email = credentials[:from_email]
  from_address = Email.new(email: from_email)
  to_address = Email.new(email: to)

  content = Content.new(type: "text/html", value:
message_body)

  mail = Mail.new(from_address, message_subject, to_
address, content)
  sg = SendGrid::API.new(api_key: api_key)
  response = sg.client.mail._("send").post(request_
body: mail.to_json)
  # try to get identifier from response body
  # convert response.status_code to our values
  return {identifier: identifier, status: status}
  end
end
```

```
#/app/services/clicksend_service.rb
class ClicksendService

  def self.send_message(method, credentials, to,
message_body, message_subject)
  username = credentials[:username]
  api_key = credentials[:api_key]
  sender_id = credentials[:sender_id]; from_email =
credentials[:from_email]

  if method == “sms”
    endpoint = “https://rest.clicksend.com/v3/sms/
send”
    body = {“messages” => [{“source” => ENV[‘APP_
NAME’], “from” => sender_id,
      “body” => message_body, “to” => to}]}.to_json

    elsif method == “email”
      endpoint = “https://api.clicksend.com/v3/email/
send”
      body = {to: to, subject: message_subject, body:
        message_body, from: from_email}.to_json
    end
      uri = URI.parse(endpoint) http = Net::HTTP.
new(uri.host, uri.port)
      http.use_ssl = true
      request = Net::HTTP::Post.new(uri.path)
      request[“Content-Type”] = “application/json”
      request.basic_auth(username, api_key)
      request.body = body
      response = http.request(request)
    # try to get identifier and status from response
body; convert status to our values
  return {identifier: identifier, status: status}
  end
end
```

```
#/app/services/mailjet_service.rb
class MailjetService
  require 'mailjet'

  def self.send_message(method, credentials, to,
message_body, message_subject)

  Mailjet.configure do |config|
    config.api_key = credentials[:api_key]
    config.secret_key = credentials[:secret_key]
    config.api_version = 'v3.1'
  end

  from_email = credentials[:from_email]
  from_name = credentials[:from_name]
  client = Mailjet::Send.create(messages: [
    {From: {Email: from_email, Name: from_name}},
    {To: [{Email: to}]},
    {Subject: message_subject},
    {HTMLPart: message_body}
  ])
  status_code = client.attributes['Messages'][0]
['status']
# convert status_code to our status values
# try to get identifier from client.
attributes['Messages'][0]
return {identifier: identifier, status: status}
end
end
```

```
#/app/services/sparkpost_service.rb
class SparkpostService

  def self.send_message(method, credentials, to,
message_body, message_subject)
  api_key = credentials[:api_key]
  from_email = credentials[:from_email]
  uri = URI.parse("https://api.sparkpost.com/api/v1/
transmissions")
  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true

  request = Net::HTTP::Post.new(uri.path)
  request['Authorization'] = api_key
  request['Content-Type'] = "application/json"
  body = {content: {from: from_email, subject:
message_subject, html: message_body, recipients:
[{address: {email: to}}]}}.to_json
    request.body = body
    response = http.request(request)
  # try to get identifier and status from response
body; convert status to our values
  return {identifier: identifier, status: status}
 end
end
```

```
#/app/services/elasticmail_service.rb
class ElasticmailService

  def self.send_message(method, credentials, to,
message_body, message_subject)
  api_key = credentials[:api_key]

  from_email = credentials[:from_email]
```

```
  uri = URI.parse('https://api.elasticemail.com/v2/
email/send')
  params = { apikey: api_key, from: from_email, to:
to, subject: message_subject,
    bodyHtml: message_body, isTransactional: true}
    http = Net::HTTP.new(uri.host, uri.port)
    http.use_ssl = true
    request = Net::HTTP::Post.new(uri.path)
    request["Content-Type"] = "application/x-www-
formurlencoded"
    request.set_form_data(params)
    response = http.request(request)
# try to get identifier and status from response body;
convert status to our values
  return {identifier: identifier, status: status}
 end
end
```

```
#/app/services/twilio-service.rb
class TwilioService
  require 'twilio-ruby'
  def self.send_message(method, credentials, to,
message_body, message_subject)
  account_sid = credentials[:account_sid]
  auth_token = credentials[:auth_token]
  from = credentials[:from]
  client = Twilio::REST::Client.new(account_id, auth_
token)
  message = client.messages.create(from: from, to: to,
body: message_body)
  sleep 2; twilio_status = client.messages(message.
sid).
  fetch.status
# convert twilio_status to our status values
```

```
    return {identifier: message.sid, status: status}
  end
end

#/app/services/plivo-services.rb
class PlivoService
  require 'plivo'
  def self.send_message(method, credentials, to,
message_body, message_subject)
  auth_id = credentials[:auth_id]
  auth_token = credentials[:auth_token]
  from_sender_id = credentials[:from_sender_id]
  client = Plivo::RestClient.new(auth_id, auth_token)
  response = client.messages.create(from_sender_
id,[to], message_body)
  identifier = response.message.uuid
# status can be retrieved with identifier

  return { identifier: identifier, status: status }
 end
end
```

```
#/app/services/smtp_service.rb
class SmtpService
  def self.send_message(method, credentials, to,
message_body, message_subject)

  ActionMailer::Base.smtp_settings = { address:
credentials[:address], port: credentials[:port],
    to: to, user_name: credentials[:user_name],
password: credentials[:password], authentication:
    credentials[:authentication].to_sym, enable_
starttls_auto: eval(credentials[:enable_starttls_
auto]), ssl:
```

```
    eval(credentials[:ssl]), tls:
eval(credentials[:tls]), verify_mode:
eval(credentials[:verify_mode]) }
    mail(to: to, from: credentials[:from_email],
subject: message_subject, body: message_body)
    return { identifier: nil, status: nil }
  end
end
```

2.9 - Administration area

As you can notice from the Client model, first registered entry will have admin privileges. Being admin, you can see the list with add clients, setup another admins or access the Sidekick web panel. Generate the clients controller with necessary functions:

```
$ bin/rails generate controller clients index show
update
```

Add the following code to the controller:

```
#/app/controllers/clients_controller.rb
class ClientsController < ApplicationController
  before_action :authenticate_client!
  before_action :check_admin_right

  def index
    @clients = Client.all
  end

  def show
    @client = Client.find(params[:id])
  end

  def update
    @client = Client.find(params[:id])
    if @client.update(client_params)
      redirect_to @client, notice: “Client updated
      successfully.”
    else
      redirect_to @client, error: “Client update
error”
    end
  end
```

```
  private

  def client_params
    params.require(:client).permit(:admin)
  end

  protected

  def check_admin_right
    redirect_to root_path, error: “You are not
authorized”
    if !current_client.admin
  end
```

Add the following routes:

```
#/config/routes.rb

resource :clients, only: [:index, :show, :update]
```

The “root” path

You have noticed many redirects to “root_path” so we have to declare it in routes file:

```
#/config/routes.rb
root “logs#index”
```

This will be the dashboard of the application, handled by “index” action of logs controller.

```
$ bin/rails generate controller logs index
```

Populate the controller with the following code:

```
#/app/controllers/logs_controller.rb
class LogsController < ApplicationController
  before_action :authenticate_client!

  def index
    @logs = Log.all
    @logs = @logs.filter_by_levels(params[:levels].
to_a) if params[:levels]
    @logs = @logs.page params[:page]
  end
end
```

Add the following pagination gem that we'll use for every resource's index action.

```
#/Gemfile
gem 'kaminari'
```

Install the gem using:

```
$ bundle install
```

Initialize the gem with the following command:

```
$ bin
```

2.10 - Project's files and folders structure

Here is a map of files handled in this book, each one with references to pages where we discussed about their code content. You can find the whole project on the Github repository at **github.com/helpliviu/secure-2fa.**

```
/app / controllers / clients / passwords_controller.rb
/registrations_controller.rb
/sessions_controller.rb
/two_factor_auths_controller.rb

/app / controllers / clients_controller.rb
/logs_controller.rb
/otp_controller.rb
/projects_controller.rb

/app / jobs / deliver_otp_job.rb
/webhook_call_job.rb

/app / models / concerns / check_linked_projects.rb

/app / models / channel.rb
/client.rb
/log.rb
/template.rb
/template_version.rb
/user.rb
/verification.rb

/app / services / clicksend_service.rb
/elastic_email_service.rb
```

```
/mailjet_service.rb
/plivo_service.rb
/postmark_service.rb
/sendgrid_service.rb
/smtp_service.rb
/sparkpost_service.rb
/twilio_service.rb

/config / initializers / devise.rb

/config / application.rb
/routes.rb
/sidekiq.yml

/db / migrations / xxx_create_projects.rb

/db / schema.db
/development.env
/Docker
/Gemfile
/Procfile
```

Chapter 3 - Using the Service

3.1 - Forever free web application

You can use the service for free at **Secure-2FA**.com. Registration is simple, requiring only a valid email address and an authenticator application (Google Authenticator or Twilio Authy).

Implementation details

You can find below the installation steps required to generate and validate an OTP. That information is also provided on each project's page inside the application.

1. Add a "require_2fa" field (type: boolean, default: true) to your "Users" table. Let the user disable 2FA from his account panel (optional).
2. In your user's authentication logic, if user's credentials are validated, save your user unique identifier (ex. id) to his session, as credentials validated and send the OTP generation request to our service using the POST call below.
3. Show the form that will validate the OTP. As the OTP validation form is submitted, your backend logic has to call the OTP verification POST below.
4. After a successful OTP validation, you can log in your user, that already has his credentials validated on step 2.

Notes

- For an OTP validation failed situation, you need to return the user to the validation form.
- It's recommended that the validation form have a link that will call the OTP generation (ex: "Re-generate code"), for expired codes or for delivery failure cases.
- After a successful login, don't forget to clear the session created before in step 2.

OTP generate call

Method: POST

Endpoint: https://opi.secure-2fa.com/otp/generate

Parameters:

REQUIRED

- encrypted_data - your user's unique identifier (uuid) + "#" + email address or phone number in E.164 international format (ex: "123#john.doe@yahoo.com")
- the entire string above will be encrypted using AES256 (examples below) with the secret_key from the project's page
- public-key - project's public key

OPTIONALS

- lang - two letters language code - first language will be selected if parameter is not sent

- ip-address - if fields are sent, will be compared with those from OTP verify call,
- user-agent - to verify an unauthorized device access
- template_request - default: false - if "true", the service will return the template populated with the generated OTP and will not send the email or SMS

OTP verify call

Method: POST

Endpoint: https://api.secure-2fa.com/otp/verify

Parameters:

REQUIRED

- encrypted_data - your user's unique identifier (uuid) + "#" + OTP six digits code
- the entire string above will be encrypted using AES256 (examples below) with the secret-key from the project's page
- public-key - project's public key

OPTIONALS

- ip-address - if fields are sent, will be compared with those from OTP generate call

- user-agent - (if present), to verify an unauthorized device access

OTP generate response

Type: JSON

Fields:

- success - true | false
- message - response description
- subject
- body - only if "template_request" value was a "true"

OTP verify response

Type: JSON

Fields:

- verified - true | false
- message - response description

Request response codes are - unauthorized (invalid or expired OTP), too-many-requests (max attempts exceeded), unauthorized_device (ip-address and user-agent do not match).

Programming languages examples
Ruby

```ruby
# required libraries
require ‘openssl’
require ‘base64’

# encryption settings
cipher = OpenSSL::Cipher::AES256.new(:CBC)
iv = cipher.random_iv

#encrypt function
def encrypt (plain_text, key, iv)
  cipher = OpenSSL::Cipher::AES256.new(:CBC)
  cipher.encrypt
  cipher.key = key
  cipher.iv = iv
  encrypted = cipher.update(plain_text) + cipher.final
  Base64.encode64(iv + encrypted)
end

#decrypt function
def decrypt(encrypted_text, key)
  decoded = Base64.decode64(encrypted_text)
  iv = decoded[0..15]
  encrypted = decoded[16..-1]
  decipher = OpenSSL::Cipher::AES256.new(:CBC)
  decipher.decrypt
  decipher.key = key
  decipher.iv = iv
  decipher.update(encrypted) + decipher.final
end
```

```
#generate OTP function
def generate_otp(encrypted_data, public_key,
lang=nil, ip_address=nil, user_agent=nil, template_
request=false)
uri = URI.parse('https://api.secure-2fa.com/otp/
generate')
params = {encrypted_data: encrypted_data, public_key:
public_key, lang: lang, ip_address: ip_address, user_
agent: user_agent, template_request: template_request}
  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true
  request = Net::HTTP::Post.new(uri.path)
  request["Content-Type"] = "application/x-www-
formurlencoded"
  request.set_form_data(params)
  response = http.request(request)
end
```

```
#verify OTP function
def verify_otp (encrypted_data, public_key, ip_
address=nil, user_agent=nil)
uri=URI.parse('https://api.secure-2fa.com/otp/verify')
params = {encrypted_data: encrypted_data, public_key:
public_key, ip_address: ip_address, user_agent: user_
agent}
  http = Net::HTTP.new(uri.host, uri.port)
  http.use_ssl = true
  request = Net::HTTP::Post.new(uri, path)
  request["Content-Type"] = "application/x-www-
formurlencoded"
  request.set_form_data(params)
  response = http.request(request)
end
```

```
#generate OTP usage
uuid = 429
to = “john.doe@yahoo.com”
data_to_encrypt = uuid.to_s + “#” + to
public_key = “…” # 82 bytes public key found on
project’s page
private_key = “…” # 32 bytes private key found on
project’s page
encrypted_data = encrypt(data_to_encrypt, private_key,
iv)
lang = “en”
ip_address = request.ip_address
user_agent = request.user_agent
generate_otp(encrypted_data, public_key, lang, ip_
address, user_agent)
```

```
#verify OTP usage
uuid = 129
otp = “123456”
data_to_encrypt = uuid.to_s + “#” + otp
public_key = “…”
private_key = “…”
encrypted_data = encrypt(data_to_encrypt, private_key,
iv)
ip_address = request.ip_address
user_agent = request.user_agent
response = verify_otp(encrypted_data, public_key, ip_
address, user_agent)
result = JSON.parse(response.body)[:verified]
```

PHP, Python, Perl, ASP.NET ….

3.2 - Open-source - Deploy in your own way

As the project is open-source, meeting all the MIT/ Apache 2.0/GPL criteria, you can clone the Github repository github.com/helplivio/secure-2fa and deploy it on premise, benefiting of data and traffic privacy. The RoR project has already a Docker file configured for a quickly deployment.

You can find implementation details in previous section **3.1**.

3.3 - Developing new features

Feel free to fork the Github repository and develop new useful and cool features. Don't forget that project is bundled with Foreman and you can start the Rails server, assets compilers and Sidekag worker with "bin/dev" command.

Add new delivery service

There are a lot of other emails or SMS delivery services that maybe fit your needs better or are more cost effective.

Here you have the steps to add a new message delivery service:

- In "Channel" model file (app/models/channel.rb - page 21), in the providers function, add a new hash entry to the providers array with the new service details. Note that email delivery services require a from_name and a from_email fields and SMS delivery services require a source-number. That fields must be included in the specific credentials array.
- Create a service file in /app/services, having the same file prefix as the service handle declared in the previous step.
- If the new service had a maintained gem, include it in Gemfile and "bundle install"it.
- Fill the service code with the same logic from existing services, replacing endpoint, request settings and parameters.
- Keep in mind that service call must return a JSON with message identifier and status. Return "nil" for both if not applicable.

Endnotes

You'll find the rest of project's files (like views, css, javascripts) on the Github repository. Considering my current situation, I decided to not document those parts of the project.

Feel free to contact me on ***helpliviu.org*** for any free assistance on any type of online security concern.

Thanks

Information for developer :)

- After registration, show a sticky top bar with message/link “Generate backup codes”. We track the visibility of this bar by client.otp-back-up-codes value, nil by default. The backup codes generation call must return an one time visible popup with a warning message to the client to save the codes. In the profile page of the client we should have the same link to re-generate the backup codes.

- In signin page, we should mention that they can use an authenticator or backup code.

- Application menu: Dashboard, Projects, Channels, Templates, Profile. Admin clients will have access also to: Clients, Sidekig.

- The dashboard, that is also the root-path must show application logs. Logs also should be present on each project’s and user’s page.

- The project’s form should include selectors for “channel” and “template” and also let client define new entries as those are nested resources.

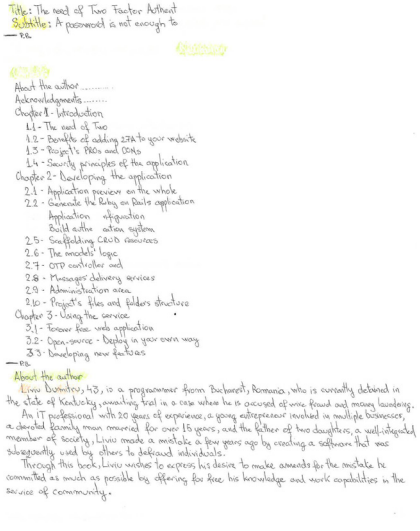
- The template show page should include all template versions. Maybe you can add a client side validation to the body to include TOTP-CODE string/var.

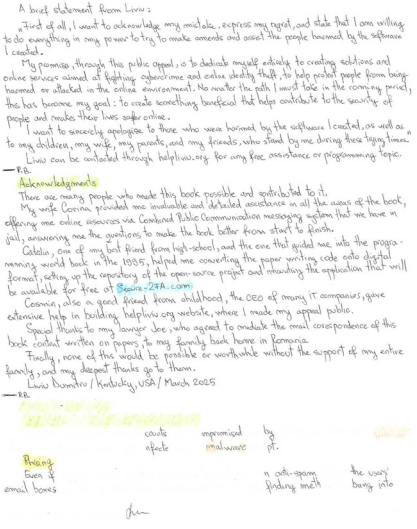
- Try to get identifier and status for each message delivery service.

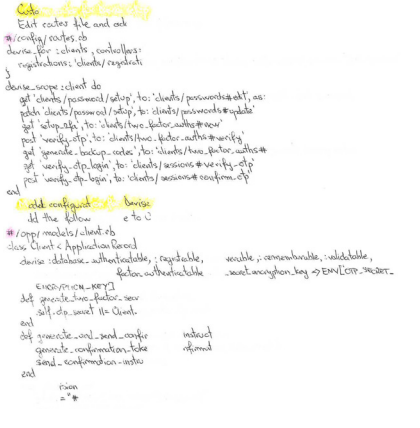
- Try to send the email also in text format.

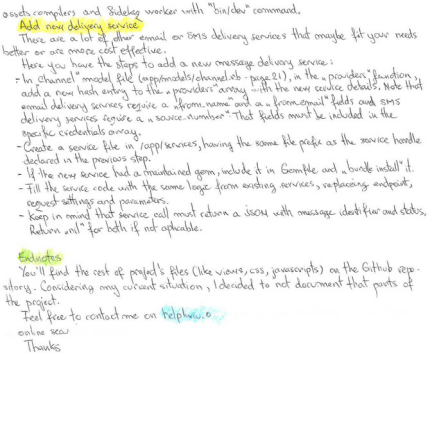
How it started

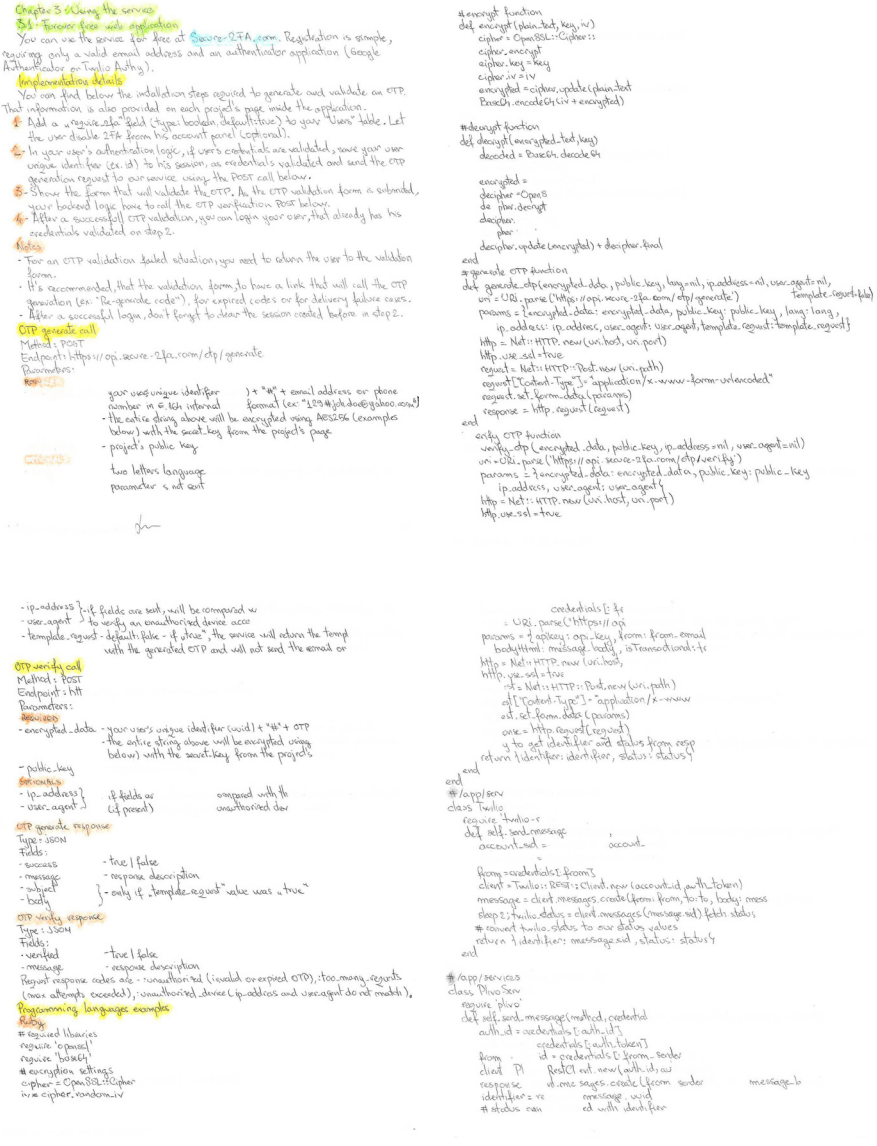
You can find here the very first form of the code—crafted line by line, by hand—offering a rare glimpse into Liviu's vision.











You can find Liviu's original handwritten continuation version of the book here: https://www.tumblr.com/blog/liviumd

The need for 2 Factor Authentication
by Liviu Dumitru

THANK YOU
helpliviu.org